Design Patterns

# Abstract Factory:

#!/usr/bin/python

from abc import ABCMeta

# Abstract Factory

class StandardFactory(object):

    @staticmethod

    def get\_factory(factory):

        if factory == 'soccer':

            return SoccerFactory()

        elif factory == 'volley':

            return VolleyFactory()

        raise TypeError('Unknown Factory.')

# Factory

class SoccerFactory(object):

    def get\_ball(self):

        return BallSoccer()

class VolleyFactory(object):

    def get\_ball(self):

        return BallVolley()

# Product Interface

class Ball(object):

    \_\_metaclass\_\_ = ABCMeta

    def play(self):

        pass

# Products

class BallSoccer(object):

    def play(self):

        return 'Ball is rolling...'

class BallVolley(object):

    def play(self):

        return 'Ball is flying!'

if \_\_name\_\_ == "\_\_main\_\_":

    factory = StandardFactory.get\_factory('volley')

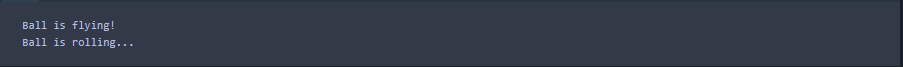
    ball = factory.get\_ball()

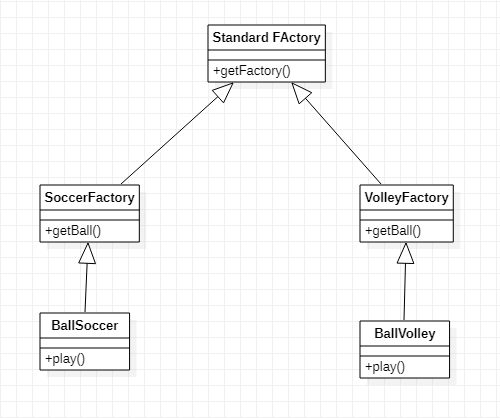
    print(ball.play())

    factory = StandardFactory.get\_factory('soccer')

    ball = factory.get\_ball()

    print(ball.play())





# Factory:

class Dog:

    """A simple dog class"""

    def \_\_init\_\_(self, name):

*self*.\_name = name

    def speak(self):

        return "Woof!"

class Cat:

    """A simple cat class"""

    def \_\_init\_\_(self, name):

*self*.\_name = name

    def speak(self):

        return "Meow!"

def get\_pet(pet="dog"):

    """The factory method"""

    pets = dict(dog=Dog("Hope"), cat=Cat("Peace"))

    return pets[pet]

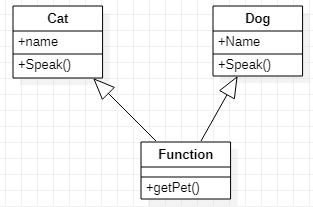
d = get\_pet("dog")

print(d.speak())

c = get\_pet("cat")

print(c.speak())





# Adapter:

d# human - Truck

# car - Car

class MotorCycle:

    """Class for MotorCycle"""

    def \_\_init\_\_(self):

*self*.name = "MotorCycle"

    def TwoWheeler(self):

        return "TwoWheeler"

class Truck:

    """Class for Truck"""

    def \_\_init\_\_(self):

*self*.name = "Truck"

    def EightWheeler(self):

        return "EightWheeler"

class Car:

    """Class for Car"""

    def \_\_init\_\_(self):

*self*.name = "Car"

    def FourWheeler(self):

        return "FourWheeler"

class Adapter:

    def \_\_init\_\_(self, obj, \*\*adapted\_methods):

        """We set the adapted methods in the object's dict"""

*self*.obj = obj

*self*.\_\_dict\_\_.update(adapted\_methods)

    def \_\_getattr\_\_(self, attr):

        """All non-adapted calls are passed to the object"""

        return getattr(*self*.obj, attr)

    def original\_dict(self):

        """Print original object dict"""

        return *self*.obj.\_\_dict\_\_

""" main method """

if \_\_name\_\_ == "\_\_main\_\_":

    """list to store objects"""

    objects = []

    motorCycle = MotorCycle()

    objects.append(Adapter(motorCycle, wheels = motorCycle.TwoWheeler))

    truck = Truck()

    objects.append(Adapter(truck, wheels = truck.EightWheeler))

    car = Car()

    objects.append(Adapter(car, wheels = car.FourWheeler))

    for obj in objects:

            print("A {0} is a {1} vehicle".format(obj.name, obj.wheels()))



